Group 2 Project 1 Design

Kyle Lehman

Anthony Ricco

Michael Norris

Steve Zogheb

# Guiding Principle

Although the assignment is for one ATM to connect to one Bank, the real world situation is an ATM can interact with many different Banks and a Bank can be connected to by many different ATMs from many different institutions. The implementation of this system will adhere to the requirement of one ATM and one Bank as outlined in the Project 1 instructions, however, the communication protocol will be designed to scale to the many ATM/many Bank model.

# Design Evolution

## Communication Encryption Protocol Selection Process

We first considered using a symmetric encryption algorithm (AES, DES, etc.) in which both the bank and ATM have built-in knowledge of the secret key. While this would have provided suitable security of the messages and would have met the strict requirements of the assignment, it did not meet the real-world requirements we have assumed in our guiding principle. It is not reasonable for every ATM to know the secret key of every banking institution with which it may need to communicate. A method for Key selection that involved an exchange of keys seemed desirable.

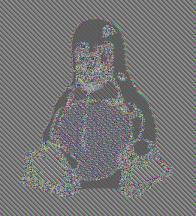
We then considered the Diffie-Hellman Key Exchange algorithm. This would allow derivation of a key through an exchange of messages in such a way as to prevent an eavesdropper from recreating the key by intercepting the handshake messages between the ATM and the Bank. Unfortunately, it does not by itself provide identity authentication. Specifically, in its unmodified form, it is vulnerable to the man-in-the-middle attack where the attacker intercepts the handshake message and pretends to be the Bank to the ATM and the ATM to the Bank. Identity authentication could have been layered onto the protocol but we decided to search further for an algorithm and protocol that provided both.

We next explored X.509 Public Key Exchange protocol but didn’t feel it was feasible to set up a full-blown X.509 Project Certificate Authority to generate the Authority’s and Bank’s Certificates in the time allotted for this project. We settled on a modified X.509-like protocol using RSA keys and encryption to verify the Bank’s identity and transfer its public key to the ATM. This provides verification of the identity of the bank in the same package that exchanges keys with the ATM. Verification of the identity of the person at the ATM is provided by the card and – most importantly – the PIN. During the course of this exchange of handshake messages, a symmetric session key is generated and exchanged between the Bank and ATM. This key and algorithm is used to encrypt all further communication for the current session. Once the session is ended, the session key is discarded and a new one is generated when the next session is initiated. Various algorithms were evaluated for the session encryption.

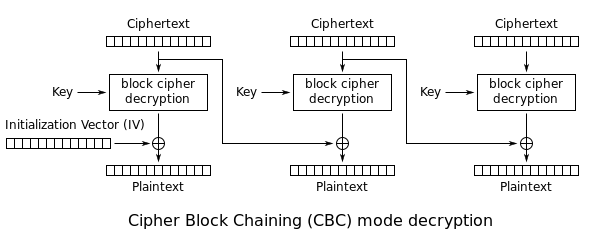
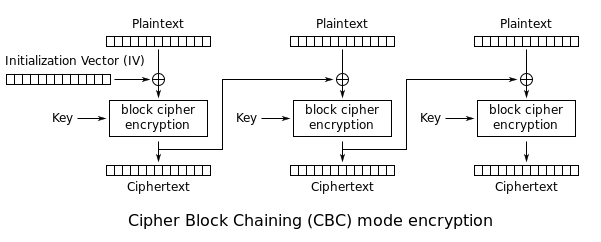
## Session Encryption Algorithm Selection

We briefly considered the Data Encryption Standard algorithm. Data Encryption Standard, or DES, was not chosen because it is now considered to be insecure for many applications, due mostly to its 56-bit key size. Next in line was the Advanced Encryption Standard algorithm, or AES. While a vulnerability was detected by researchers in 2011, “the number of steps required to crack AES-128 is an 8 followed by 37 zeroes. To put this into perspective: on a trillion machines, that each could test a billion keys per second, it would take more than two billion years to recover an AES-128 key,” and “that a ‘practical’ AES crack is still far off”. Therefore, the Advanced Encryption Standard algorithm, or AES, was chosen for the symmetric key. Due to compatibility reasons the key size defaults to 128 bit although support for 256 keys can be enabled with a configuration file change. [<http://www.theinquirer.net/inquirer/news/2102435/aes-encryption-cracked>]

The next choice was to decide what mode of encryption to use, with the choices being Electronic Code Book (ECB), or Cipher Block Chaining (CBC). In ECB mode the message is divided into blocks and each block is encrypted separately. The disadvantage of this method is that identical plaintext blocks are encrypted into identical ciphertext blocks; which means it does not hide data patterns well. An excellent example of this is the encryption of bitmap images (please see the images below). While the color of each pixel is encrypted, the overall image is still discernible. [<https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>]



The next choice is CBC where each block of plaintext is XORed with the previous ciphertext block before being encrypted. That means each ciphertext block depends on all plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block, as shown in the following figure: [<https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation>]



# Security Protocol Implementation

## Certificate Authority

We first developed a Certificate Authority application (CA.java) to generate the Bank certificate and RSA key pairs for the Bank and the Certificate Authority (CA). All keys are 4096 bits long. The public key of the Project Certificate Authority acts as the trusted root certificate installed in the ATM. The ATM in this application will have only one trusted root certificate installed. After generation of the initial key pairs, the Bank Certificate was generated. The keys and certificate are stored in files as serialized objects (G2\_BANK\_Certificate.ser, G2\_BANK\_PrivateKey.ser, and G2\_CA\_PublicKey.ser). At runtime, the Bank will know its certificate and private key. The ATM will only know the public key of the Certificate Authority. The certificate is stored in a Java [SignedObject](http://docs.oracle.com/javase/7/docs/api/java/security/SignedObject.html) class using the CA’s private RSA key. The Signature algorithm is SHA512withRSA. Validation of a certificate is described in the next section.

## Certificate Validation

Our G2MX.509 (Group 2 Modified X.509) certificate format is as follows…

A certificate is valid if it passes the following checks.

1. Using the root CA’s public key (which the ATM implicitly knows), the verify(…) method of the SignedObject returns true.
2. The CA name in the certificate must match the CA name that the ATM implicitly knows.
3. The CA Public Key in the certificate must match the root CA Public Key stored in the ATM.
4. The Bank Name in the certificate must match the name of the Bank the ATM is attempting to communicate with.

## Session Handshake

When the ATM wishes to initiate a session, it first sends an initiate key exchange message. This message is sent in the clear. The Bank responds with its certificate – contained within a SignedObject - as defined above in G2MX.509 Certificate Structure and a unique Session Id for this session. If the Certificate is valid, the ATM saves the session id, generates a session 128-bit (optional 256-bit) AES key, and encrypts the key along with the card/account information and pin into a Java SealedObject using the RSA 4096-bit public key of the Bank which it obtained from the certificate. Note that RSA encryption and decryption uses "RSA/ECB/PKCS1Padding" as that combination is supported by all Java implementations. The bank decodes the message with its private key, validates that the Session Id matches the one it sent with the certificate, validates the user account and pin information and saves the secret key for all future communication under that session. It then sends a confirmation message to the ATM. That message will either inform the ATM that the user information is valid - the ATM can proceed to send session messages - or that the user information is invalid and the session should be terminated. The Handshake message sequence is depicted graphically below.

## Session Communication

Once the handshake sequence is over, normal session communication can commence. In this phase, no further sensitive user credentials or Bank certificate need to be exchanged. It is worth restating that all sensitive user credentials are encrypted using the strong RSA 4096 bit public/private keys of the bank. After the handshake exchange of credentials, the integrity and security of the remaining session communication is maintained by the session id, secret key, AES encryption and bank account number. The bank uses this information to validate all communication from the ATM and verifies that only the proper accounts are accessed.

A critical part of the session security is the session’s key and symmetric cipher algorithm. The key is 128-bit AES and the cipher algorithm is AES/CBC/PKCS5Padding. Additionally, every message uses a unique Initialization Vector (IV). The key, algorithm and IV are used to encrypt the message into a Java [SealedObject](http://docs.oracle.com/javase/7/docs/api/javax/crypto/SealedObject.html). The unique IV is created for each message using a Java SecureRandom object that uses the “SHA1PRNG” pseudo-random number generation algorithm and ensures that two instances of the same transaction will generate a unique encrypted cipher text. This prevents ciphertext analysis of successive encryptions of the same message from compromising the secret key. It should also be noted that although the default AES key size is 128, the application supports 256 bit keys with a property file change and the installation of the Java “Unlimited Strength Jurisdiction Policy” update to the Java Cryptographic Extension (JCE). This update is simply to copy two Java supplied jar files into the lib/security directory under the java runtime home directory.

In addition to the session encryption described above, other measures are taken by the application to enhance the security of the session transactions. It has already been mentioned that the Bank looks at each transaction and makes sure that it maps to an active session and that the account information matches the account of the associated user. This prevents crossover attacks in which the attacker uses the card and pin of one user to withdraw from the account of another user. Another security measure is a 3-strikes-you’re-out policy. In this policy, the bank institutes a one hour lockout of any account after three consecutive attempts to access an account with an invalid pin. This discourages attempts to brute-force compromise the pin by making it take over 138 days to cycle through and try all 10000 possible pins. The one hour lockout is configurable through a property file on the bank side. Another possible attack avenue is to compromise the AES session key. We think this attack is unlikely to succeed but, if it were to happen while the session was active a man-in-the-middle attacker could send its own encrypted transactions to the bank and withdraw money from the user’s account. To deal with this type of attack, the bank enforces a strict session timeout of one minute. After one minute the bank unilaterally terminates the session. The ATM learns of this when it next attempts to send a message to the bank using the now invalid session id. So the window of vulnerability in which an attacker can use a compromised session key is reduced to one minute. If the key can be broken in a minute, the attacker gains access to the user’s account. After the minute has expired, a broken session key only the user’s privacy is compromised; his account activity and/or account balance. That is unfortunate but he can rest assured that his life’s savings remain secure.

# Conclusion

We feel this solution answers the requirements of the assignment. The trusted root certificate authority, signed bank certificate validated by the ATM, RSA key exchange, RSA encryption of the user’s pin and card information, use of an AES session key with unique message initialization vectors, bank validation of each session transaction, 3-strikes-you’re-out rule and session timeout work together to build a formidable security barrier confronting a potential attacker. It does this while providing an infrastructure that can scale to easily incorporate additional ATMs and Banks into the system. We will conclude by briefly mentioning a type of attack that this system does not address. It does not address various denial-of-service attacks. One such attack would be to flood the bank with bogus messages. Another would be to interdict the message traffic between the ATM and Bank. A final would be to intercept the message traffic and reorder or scramble the message before forwarding it to the Bank. While our solution cannot prevent this type of attack, it should not be compromised by this kind of attack. At best, this attack will prevent transactions from completing successfully but should in no way allow the attacker to obtain access to the user’s account. Therefore, we feel this solution meets the requirements of the assignment, including the additional ones we added in the first paragraph of this paper.